A Toolbox for Counter-Example Analysis and Optimization

Alan Mishchenko Niklas Een Robert Brayton

Department of EECS, University of California, Berkeley

{alanmi, een, brayton}@eecs.berkeley.edu

Abstract

Counter-examples are produced by formal verification engines to witness failures of safety properties. A counter-example is a sequence of input assignments bringing the design from the initial state into a state where some property fails. In practice, these input assignments contain redundancies. This paper focuses on methods for analyzing counter-examples to detect don't-care, optional, and essential input assignments. The proposed analysis of counter-examples helps design debugging. Additionally, it is useful to reduce the length of counter-examples derived by random simulation and for efficient refinement in localization abstraction.

1. Introduction

Formal verification is often applied to hardware designs in order to check safety properties, in particular, design equivalence. When a particular property fails, a counter-example (CE) is produced. The CE can be simulated on the design to reproduce the property failure.

In a typical design verification environment, CEs are essential for debugging hardware designs. This motivates analysis and optimization of CEs to make them more explicit (containing only relevant information) or shorter (taking fewer cycles from the initial state to the property failure).

In this paper, we concentrate on the first task - making CEs more explicit. The goal is to facilitate design debugging, by allowing verification engineers to focus on the immediate cause of a failure, without having to sort through irrelevant information. Experiments show that typical CEs contain only 5-10% of relevant information.

The secondary goal of this work is to gather information needed for (a) making counter-examples shorter and (b) making CE-based abstraction more efficient. Although it is clear that the proposed analysis can be useful in these applications, experimentation is deferred to future work.

The rest of the paper is organized as follows:
Section 2 provides motivation for CE analysis.
Section 3 contains necessary background.
Section 4 introduces CE-induced network.
Section 5 presents algorithms used.
Section 6 shows experimental results.
Finally, Section 7 concludes the paper.

2. Motivation

To motivate the need for CE analysis and optimization, consider a typical verification scenario:

A verification engine, such as a SAT-based Bounded Model Checker [2], guided random simulator [8], or IC3/PDR [4][7], produces a CE containing a complete assignment of all primary inputs in all timeframes. Thus if the design with N primary inputs fails a safety property in frame F, the witnessing CE contains $M = N * F$ bits, one for each input in each timeframe.

Suppose a subset of these M bits is isolated and their values are kept unchanged while the values of other bits are changed, resulting in a new CE. Whether the new CE fails the property in frame F depends on what bits are modified.

This paper answers this question by dividing the set of all bits of a CE into three categories:

- *don't-care bits:* these can be changed, one at a time or all at once, and the modified CE still fails the property;
- *essential bits:* these bits cannot be modified without affecting the capability of the CE to fail the property;
- *optional bits:* these bits can be modified under some conditions (for example, if we change one of them, we need to change another, in order to preserve the fact that the modified CE still fails the property).

Example: Consider circuit F = OR(a, AND(b, c), AND(b, d)) and input assignment (a, b, c, d) = (0, 1, 1, 1). Assume that value 1 produced at the output denotes the failure of some property. The input variables a, b, c, and d can be classified as follows: a is a don't-care bit because its value does not impact the output value; b is an essential bit because value 1 at the output cannot be produced without b being 1; finally, c and d are optional because any of the two combinations, (b, c) = (1, 1) and (b, d) = (1, 1), are sufficient to produce value 1 at the output.

Note that this example shows that the type of a bit is relative to the bit values given by the CE. Thus $b = 1$ is not essential if we make $a = 1$. The latter is a different CE.

Below we propose algorithms to compute bit types for all bits of an arbitrary CE derived by a verification engine.

We also propose a heuristic way of computing an optimized CE that fails the property while the number of its care bits (essential and optional bits) is minimized.

The proposed algorithms do not use BDDs, SAT or ternary simulation. Instead, they are based on guided simulation and exploring the circuit structure, and are therefore more scalable.

3. Background

3.1 Boolean network

A *Boolean network* (or *circuit*) is a directed acyclic graph (DAG) with nodes corresponding to logic gates and edges corresponding to wires connecting the nodes. It is assumed that each node has a unique integer called *node ID*.

A node *n* has zero or more fanins, i.e. nodes driving *n*, and zero or more fanouts, i.e. nodes driven by *n*. The primary inputs (PIs) are nodes without fanins in the current network. The primary outputs (POs) are a subset of nodes of the network, connecting it to the environment. A fanin (fanout) cone of node *n* is a subset of nodes of the network, reachable through the fanin (fanout) edges of the node.

A *sequential* network contains registers which store values of the state bits in the current timeframe. Register inputs and outputs are often treated as additional POs/PIs of the combinational logic of the design.

In formal verification, *a sequential miter* is a sequential Boolean network representing the design together with property monitors expressed using (possibly sequential) logic cones. The outputs of these monitors represent the properties to be proved. A typical sequential miter has an *initial state*, which is an assignment of values (0, 1, or "unknown") to the registers in the starting timeframe.

3.2 And-Inverter Graph

And-Inverter Graph (AIG) is a Boolean network whose nodes can be classified as follows:

- One constant 0 node.
- Combinational inputs (primary inputs, flop outputs).
- Internal two-input AND nodes.
- Combinational outputs (primary outputs, flop inputs).

The fanins of internal AND nodes and combinational outputs can be complemented. The complemented attribute is represented as a bit mark on a fanin edge, rather than a separate single-input node.

Due to their compactness and homogeneity, AIGs have become a de-facto standard for representing sequential miters in formal verification.

An *AIG manager* is a data-structure used to store AIGs in a software implementation of verification engines. A typical AIG manager enforces the following invariants during AIG construction and manipulation:

- Constants are propagated (the constant 0 can only feed into a combinational output).
- Duplicated fanins of AND nodes are not allowed.
- Fanins of AND nodes are ordered using their node IDs.
- AND nodes are structurally hashed (each AND node has a unique pair of fanins, possibly complemented).
- All AIG nodes are stored in a topological order (that is, fanins of each node precede the node itself).

3.3 Counter-examples

A counter-example (CE) for a safety model checking problem is characterized by the following information relevant to the presentation in this paper:

- The sequential miter with initial state.
- The number of a primary output that failed.
- The number of transitions (timeframes) to the failure (also called the *depth* of the CE).
- Binary values (0 or 1) assigned to each input of the miter in each timeframe (also called the *bits* of the CE).

Note that, in this paper, a CE is defined to be independent of the initial state information. This information is irrelevant for the proposed analysis of CEs and is considered part of the sequential miter.

A CE can be validated by initializing the sequential miter and simulating the bits of the CE assigned to the primary inputs for the depth indicated by the CE. If the specified output fails in the last timeframe of the CE, the CE is *valid*. Otherwise, the CE is *invalid*. An invalid CE can indicate a software bug in the verification engine. In the proposed work, invalid CEs can arise as intermediate steps during CE optimization.

3.4 Justifying subset

A subset of bits of the CE is called a *justifying subset* if preserving the values of these bits and arbitrarily modifying the values of other bits, results in a valid CE. Conversely, a subset is called *non-justifying*, if it results in an invalid CE. Intuitively, the values of the bits belonging to a justifying subset form a condition sufficient to imply the property failure in the last timeframe, as mandated by the CE.

A justifying subset is *minimal* if removing any bit from it results in an invalid CE.

Now we re-state classification of CE bits from Section 2 using the definition of the justifying subset:

- *a don't-care bit* can be removed from a justifying subset resulting in another (smaller) justifying subset (in other words, minimal justifying assignments do not contain don't-care bits);
- *an essential bit* is included in *any* justifying subset (in other words, excluding an essential bit from a justifying subset makes it a non-justifying subset);
- *an optional bit* belongs to at least one minimal justifying subset and may be missing in some other justifying subset.

4. CE-induced network

This section presents the main technical novelty of the proposed work, the notion of a *network induced by a CE*.

Consider a sequential miter and a valid CE of this miter. Unroll the miter for the depth of the CE. Validate the CE by initializing the miter and propagating the value of the bits of the CE assigned to the primary inputs of the miter, to the last timeframe where the property fails. This procedure assigns a binary value to each object of the unrolled miter.

Furthermore, assume that a sequential miter and its unrolling are AIGs. The unrolled AND nodes have values assigned to them based on a CE, as discussed above.

Consider a combinational network, called *CE-induced network*, with the same primary inputs as the unrolled AIG. This network is constructed using two-input AND/OR nodes and buffers as follows:

1) if the output of an AND node of the unrolled AIG has value 1 under the CE, replace it by an AND in the CE-induced network;

2) if the output of an AND node of the unrolled AIG has value 0 and both fanins have value 0, replace it by an OR in the CE-induced network;
3) if the output of an AND node has value 0 and only one input has value 0, replace it by a buffer fed by the fanin which has the controlling value 0. The other fanin is left dangling.
4) nodes that have no path to the output are removed.

The resulting network has only one PO corresponding to the property output failed by the CE and possibly many primary inputs without fanout.

Important properties of the CE-induced network are:

- it is unique for the given CE, because the CE forces unique values on the unrolled nodes and the rules used to construct the CE-induced network have no ambiguities;
- it is a *positive unate* Boolean function in terms of its primary inputs, which is composed of buffers, AND nodes and OR nodes, without inverters.

The inverters appearing at the fanins of some nodes in the unrolled AIG are ignored because the CE-induced network is tracking the dependency of a signal on its fanins, irrespective of the phase of the signal**.**

Theorem 1: *The CE-induced network is the characteristic function of all justifying subsets of the CE.*

Proof: A justifying subset is a subset of bits that produces a 1 at the output of the miter, independent of the values of the other bits. The output of the miter can only become 1 if it has 1 on at least one of its inputs if the gate is an OR, or both of its inputs are 1 if it is an AND. This continues through all gates having 1s at their outputs until the primary inputs are reached. By construction, these paths of 1s in the CE-induced network correspond to justifying paths for the CE of the original network. As such, if the bit-values of the CE of the subset corresponding to the inputs of the 1-paths of the CE-induced network are kept the same as in the CE, the output of the network is 1 independent of other input values. Hence this subset is justifying for the CE. **QED**

Our experiments indicate that the CE-induced network can be easily constructed and its size is typically an order of magnitude smaller than the size of the bounded unrolling of the sequential miter for the depth indicated by the CE. The CE-induced network is highly redundant and lends itself to efficient optimization by logic synthesis.

Although the actual CE-induced network has several important applications, to be discussed later, in many cases it does not have to be constructed explicitly.

In particular, the algorithms presented in this paper have been implemented without explicitly deriving the CE-induced network. The unrolling of the miter and the traversal of the CE-induced network are performed conceptually, by repeatedly traversing a single timeframe of the miter. This results in substantial memory and runtime savings, compared to an implementation that derives the unrolling and the CE-induced network explicitly.

5. Algorithms

As stated in Section 2 and re-stated in Section 3.4, the bits of a CE can be divided into three disjoint classes: (a) don't-care bits, (b) essential bits, (c) optional bits. In this section, we show how to compute them for a given CE.

5.1 Classifying bit types of a CE

Don't-care bits

The computation of don't-care bits is straight-forward. Consider the CE-induced network and find its primary inputs without fanout. These primary inputs do not particulate in minimal subsets and are, therefore, don't-care bits of the CE. Any input that has a path to the output is not a don't care because there is a controlling path, induced by the CE, to the output of the network.

Essential bits

Essential bits are those that participate in all justifying subsets. Since the CE-induced network is the characteristic function $F(x)$ of all justifying subsets x, the essential bits are primary inputs x_i, such that $F|_{x=0} = 0$.

Since F is positive unate, we can check if x_i is essential by simulating the CE-induced network with the PI combination composed all 1s with only one 0 for variable x_i. The corresponding bit of the CE is essential iff the simulation produces 0 at the output of the CE-induced network.

In this algorithm for detecting essential bits, each care CE bit requires a separate simulation round. However, bit-parallel simulation can be used to simulate multiple patterns at the same time. Moreover, most of the simulation patterns result in simulation traces identical to those of some other patterns. This can be exploited to compress intermediate simulation patterns, resulting in a reduced runtime.

Optional bits

Optional bits of a CE are those that are neither don't-care nor essential bits of the CE.

5.2 Minimizing a justifying subset

To compute a heuristically minimized justifying subset of CE bits, perform the following steps:

- Assign the values of all AIG nodes of the unrolled sequential miter according to the CE.
- Propagate implications due to (a) constant 0 nodes, (b) the initial state of the design and (c) the values of the essential bits. Mark all the internal nodes of the unrolled miter whose value is fixed due to these implications.
- Traverse the unrolled miter in the reverse topological order starting from the failed PO and stop at the nodes whose value is already implied, as shown above. If the value of an AND node is not implied, do as follows:
  - if the node has value 1, traverse both fanins;
  - else if the node has value 0 and both fanins have value 0, traverse only one of them;
  - else if the node has value 0, traverse the fanin whose value is 0.

When this traversal is completed, some primary inputs have been reached. These form a justifying subset. Although, in general, this subset is not minimal, we

conjecture that it is generally close to minimal and plan to conduct experiments to verify this as part of future work.

Note that this traversal is different from the one used to construct the CE-induced network in Section 4. The difference is that, when a node and both of its fanins have value 0, we traverse *only one* of the fanin (rather than both).

Another important aspect of the above procedure is that it begins by computing implied nodes and bypasses them when choosing a justifying subset. This way, primary inputs cut-off from the output of the miter by the propagation of the initial state or by the essential bits, are not included in the justifying subset under construction.

5.3 CE validation with a justifying subset

The usual CE validation *without* using a justifying subset is performed by simulating the values of the CE bits through the unrolled miter and making sure that the property fails in the last timeframe, as stated by the CE.

To validate a CE with a (minimal) justifying subset one can use ternary simulation. The bits included in the justifying subset are set according to the CE, while the remaining bits are assumed to have ternary value "X". Then the unrolled miter is simulated using the standard rules of ternary simulation [6].

Theorem 2: *If the property output fails under ternary simulation in the last timeframe then the subset is justifying (that is, the corresponding CE is valid).*

Proof. According to the rules of ternary simulation, the output of an AND or OR with an X on an input is not X if the other input is controlling. The fact that the property output is not X under ternary simulation means that the output value is controlled by 0s and 1s implied by the primary input bits of the justifying subset. This is similar to ordinary simulation and hence the subset is justifying. **QED**

Note that the converse of the theorem is not true because ternary simulation is conservative, i.e. if the output is X, it does not mean that the ordinary simulation would not produce 1 (property fails). However, as yet we have not encountered such an outcome.

6. Experimental results

The proposed algorithms for computing don't-care, optional, and essential bits of a CE are implemented in a command *&cexinfo* in a recent version of ABC [1].

The input to the command is the current AIG and the current CE for this AIG generated by one of the verification engines in ABC. The output is the printout of statistics regarding the ratio of bits of each type. A minimized justifying subset of the CE is accessible for applications calling ABC as a static library via programmable APIs.

The benchmarks are satisfying testcases from Hardware Model Checking Competition 2012 [3] solved by ABC [5]. The testcases were preprocessed by sequential signal correspondence (command "scorr") [9]. Next three engines were applied: BMC (command "bmc3"), IC3/PDR (command "pdr"), and rarity-based simulation (command "sim3") with timeouts 100 sec. The resulting AIG/CE combinations (if the CE was found and its length exceeded 10) were used to test the proposed algorithms.

The CE analysis results are shown in Table 1. The first column gives the name of a testcase. The second column is the name of the engine that produced the CE. Listed next are the number of primary inputs and flop-flops after the preprocessing, and the number of timeframes in the CE.

The CE bit data includes the following: The total number of input bits and the percentages of don't-care, optional, essential, and minimized bits. CEs with minimized justifying assignments were validated using ternary simulation, as described in Section 5.5.

The runtimes listed in the final column of Table 1 is the cumulative runtime taken by the algorithms to classify the bits, find a minimized justifying subset, and validate it using ternary simulation. The runtime is dominated by the computation of essential bits. We believe that a better implementation can substantially reduce this runtime.

It can be noted that CEs generated by random simulation take a longer time to analyze. This gives additional motivation to develop efficient methods for CE depth minimization.

Table 2 compares the sizes of two combinational circuits: (1) the initialized unrolling of sequential miters included in Table 1 for the depth specified by the CE, and (2) the CE-induced network for the same CE. Only the bounded cone of the PO of the last time is included in the unrolling of sequential miters, to make it comparable to the CE-induced network having a single PO at the same depth.

It can be seen from Table 2 that the PI counts are the same, as expected, while the number of AND nodes (levels) in the CE-induced networks is, on average, about 5X (3X) smaller compared to the unrolling of sequential miters.

Not shown in Table 2 is the observation that CE-induced networks have more fanoutless PIs and are more readily reduced by logic synthesis, compared to unrolled miters.

7. Conclusions and future work

This paper focuses on analyzing and optimizating counter-examples (CEs) produced by formal verification engines. The concept of a CE-induced network was introduced for efficient computation of don't-care, essential, and optional bits of a CE. An algorithm to compute a minimized justifying subset of a CE is proposed.

We believe that this work can facilitate design debugging by giving verification engineers information that makes it easier to understand the reason of a property failure.

We also point out that the analysis applied in this paper to the input values of a CE can be extended to cover the state bits of the internal states generated by a CE in the intermediate timeframes. Don't-care, essential, and optional bits of these states can be computed and used for various CE transformations. Also, the CE-induced network introduced and used conceptually in this paper, can be viewed as a symbolic representation of all justifying subsets. Because the size of this network is substantially smaller than the size of the bounded unrolling of the miter, it can be used to compactly represent sets of the states, for which the trace to the property failure is known.

Future work will concentrate on developing methods for CE depth minimization and improving abstraction refinement using the notion of the CE-induced network.

8. REFERENCES

[1] Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification*. http://www-cad.eecs.berkeley.edu/~alanmi/abc

[2] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without BDDs". *Proc. TACAS'99*, pp. 193-207.

[3] A. Biere, HWMCC 2012, http://fmv.jku.at/hwmcc12/

[4] A. R. Bradley. "SAT-based model checking without unrolling". *Proc. VMCAI'11*.

[5] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool", *Proc. CAV'10*, LNCS 6174, pp. 24-40.

[6] R. Bryant. "Boolean analysis of MOS circuits". *IEEE Trans. CAD*, July 1987, pp. 634-649.

[7] N. Een, A. Mishchenko and R. Brayton, "Efficient implementation of property-directed reachability", *Proc. FMCAD'11*.

[8] R. Brayton, N. Een, and A. Mishchenko, "Using speculation for sequential equivalence checking", *Proc. IWLS'12*, pp. 139-145.

[9] A. Mishchenko, M. L. Case, R. K. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis", Proc. ICCAD'08, pp. 234-241.

Table 1: Ratios of don't-care, optional, essential, and minimized bits in CEs produced by verification engines.

Example	Engine	PIs	FFs	Frames	Total bits	DC, %	Opt, %	Essen, %	Min, %	Time, s
6s1	SIM	45	291	1247	56115	53.95	27.50	18.55	29.72	173.33
6s5	SIM	141	2519	63	8883	60.32	37.95	1.73	25.69	11.48
6s14	SIM	439	811	869	381491	83.16	15.25	1.59	4.38	141.72
6s17	SIM	450	819	1084	487800	83.50	15.23	1.27	4.43	213.70
6s18	SIM	450	819	496	223200	83.14	15.41	1.44	4.56	83.53
6s133	SIM	450	819	1084	487800	83.50	15.23	1.27	4.44	209.30
6s134	SIM	36	571	610	21960	90.83	7.02	2.15	7.19	3.83
bob12s05	SIM	437	3956	43	18791	65.60	31.48	2.92	30.09	28.44
bobtuttt	SIM	2807	111	1582	4440674	98.74	1.24	0.02	0.09	57.50
6s41	BMC	19	959	74	1406	58.46	33.57	7.97	23.26	0.61
6s134	BMC	36	571	169	6084	92.11	5.34	2.55	5.85	0.59
6s162	BMC	73	156	74	5402	79.67	13.16	7.16	13.92	0.42
6s172	BMC	403	422	46	18538	66.71	29.48	3.81	10.42	1.48
6s172	PDR	403	422	111	44733	47.06	49.10	3.83	17.34	7.70
Geo					1.000	0.731	0.157	0.021	0.076	

Table 2: Comparison of AIGs representing BMC unrolling of the miter (left) and CE-induced network (right).

Example	CE Depth	PI	AND	Level	Time, s	PI	AND	Level	Time, s
6s1	1247	56115	3514473	43546	1.91	56115	575536	9243	0.63
6s5	63	8883	1571421	2595	0.55	8883	548554	1596	0.38
6s14	869	381491	12102021	318775	6.66	381491	2666982	112934	5.78
6s17	1084	487800	12906342	330886	7.31	487800	2925648	114392	6.05
6s18	496	223200	5572818	151699	3.17	223200	1292920	53502	2.33
6s133	1084	487800	12906261	330885	7.16	487800	2926433	114394	6.14
6s134	610	21960	103586	6430	0.09	21960	18033	2492	0.06
bob12s05	43	18791	607028	743	0.38	18791	196375	548	0.30
bobtuttt	1582	4440674	14682128	64774	6.30	4440674	369044	13142	3.34
6s41	74	1406	103884	549	0.05	1406	27600	259	0.02
6s134	169	6084	21844	1495	0.02	6084	3925	639	0.02
6s162	74	5402	67990	1666	0.02	5402	16427	477	0.06
6s172	46	18538	120433	822	0.03	18538	45363	484	0.01
6s172	111	44733	389403	2382	0.11	44733	153616	1472	0.08
Geo		1.000	1.000	1.000	1.000	1.000	0.212	0.397	0.711